21-1/ macros-by-example

```
(define-syntax-rules (let*
  [(_ () body)                          <- pattern
   (begin body)]                        <- template
  [(_ ([x xe] more ...) body)
   ((λ (x) (let* (more ...) body)) xe)])
```

```
(let* ([x 5] [y (+ x 6)]) (+ x y))
```

macro-env : id → macro-def = list (pat * tmp)

macro-apply : macro-def × macro-invocation

1. for each pat & template
   a. does the invoke match the pattern?

match: stx invoke [_° → let*, x° → x, xe° → 5,
→ #f or env more' → ([y (+ x 6)]), body° → (+ x y)]

[id, level] → stx

   b. transcribe : template × match-env → stx

((λ (x) (let* ([y (+ x 6)]) (+ x y))) 5)

   c. done

21-2 (match e [pat body] ...)

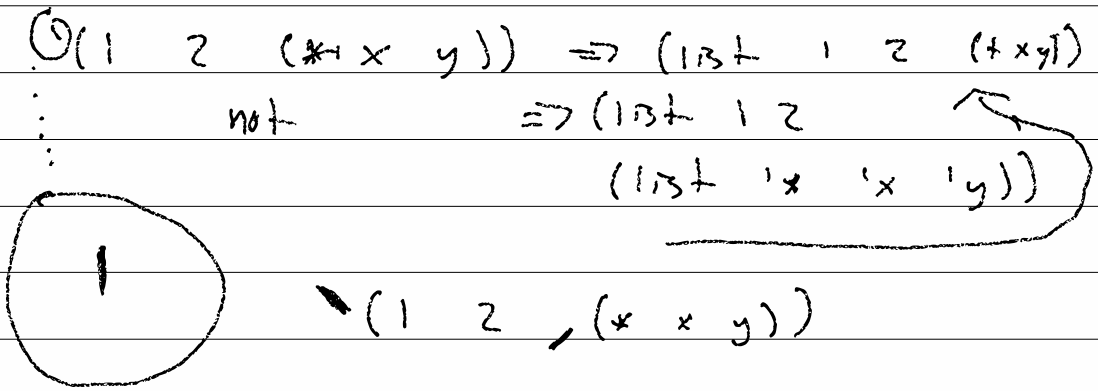
```
(match x
  ['()  5]
  [(cons a b)  (+ a b)])
        =>
(cond
  [(empty? x)  5]
  [(cons? x)
   (let ([a (first x)] [b (rest x)])
     (+ a b))]
  [else (error "no case for: ~e" x)])
```

```
[(cons A (cons B (cons C '()))) X]
[(list A B C)                   X]
```

```
'(1 2 3) => (list 1 2 3)
 => (quote (1 2 3))
```

21-3

```
(dsrs quote
  [(a ...)  (list (quote a) ...)]
  [(NUMBER n)  n]
  [()  empty])
```




```
(dsrs quasiquote (unquote)
  [(unquote a)  a])
```

match

```
['(1 2 3)  ->  (and (list? x)
                    (= 3 (length x))
                    (= 1 (1st x))
                    (= 2 (snd x))
                    (= 3 (third x)))]

[`(1 2 ,x)  ==>
   (let ([x (3rd x)])
     ...)
```